Editor: Dianne P. O'Leary, oleary@cs.umd.edu

FINITE DIFFERENCES AND FINITE ELEMENTS: GETTING TO KNOW YOU

By Dianne P. O'Leary

NUMERICAL SOLUTION OF DIFFERENTIAL EQUATIONS RELIES ON TWO MAIN METHODS: FINITE DIFFERENCES AND FINITE ELEMENTS. IN THIS HOMEWORK ASSIGNMENT, WE explore the nuts and bolts of the two methods for a simple *two-point boundary value problem*:

$$-(a(x)u'(x))' + c(x)u(x) = f(x) \text{ for } x \in (0, 1),$$

with the functions a, c, and f given, and $u(0) = u(1) = 0$.

We will assume that $a(x) \geq a_0$, where a_0 is a positive number, and $c(x) \geq 0$ for $x \in [0, 1]$.

The Finite Difference Method

Let's rewrite our equation as

$$-a(x)u''(x) - a'(x)u'(x) + c(x)u(x) = f(x) \quad (1)$$

and approximate each derivative of u by a finite difference:

$$u'(x) = \frac{u(x) - u(x-h)}{h} + O(h),$$

$$u''(x) = \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} + O(h^2).$$

(We'll compute $a'(x)$ analytically, so we won't need an approximation to it.)

The finite difference approach is to choose *mesh points* x_j

Tools

To debug your programs, it's helpful to experiment with the simplest test problem and a small number of mesh points. Look ahead to Problem 6 for sample problems.

Problem 2 uses the Matlab function `spdiags` to construct a sparse matrix. If you have never used sparse matrices in Matlab, print the matrix `A` to see that its data structure contains the row index, column index, and value for each nonzero element. If you have never used `spdiags`, type `help spdiags` to see the documentation, and then try it on your own data to see exactly how the matrix elements are defined.

Use Matlab's `quad` to compute the integrals for the entries in the matrix and right-hand side for the finite element formulations.

Before tackling the programming for Problems 5 and 6, take some time to understand exactly where the nonzeros are in the matrix, and exactly what intervals of integration should be used. The programs are short, but it's easy to make mistakes if you don't understand what they compute.

In Problem 7, we measure *work* by counting the number of multiplications. One alternative is to count the number of floating-point computations, but this usually gives a count of about twice the number of multiplications, because multiplications and additions are typically paired in computations. Computing time is another very useful measure of work, but it can be contaminated by the effects of other users or computer processes.

In determining and understanding the convergence rate in Problem 7, plotting the solutions or the error norms might be helpful.

Mark Gockenbach gives a good introduction to the theory of finite difference and finite element methods;[1] for a more advanced treatment, see, for example, Stig Larsson and Vidar Thomée's book.[2]

References

1. M.S. Gockenbach, *Partial Differential Equations: Analytical and Numerical Methods*, SIAM Press, 2002.
2. S. Larsson and V. Thomée, *Partial Differential Equations with Numerical Methods*, Springer, 2003, Chapter 2, Sections 4.1 and 5.1.

 Copublished by the IEEE CS and the AIP 1521-9615/05/$20.00 © 2005 IEEE

$= jh$, where $h = 1/(M-1)$ for some large integer M, and then solve for $u_j \approx u(x_j)$ for $j = 1, \ldots, M-2$. We write one equation for each unknown, by substituting our finite difference approximations for u'' and u' into Equation 1, and then evaluating the equation at $x = x_j$.

Problem 1.

Let $M = 6$, $a(x) = 1$, and $c(x) = 0$ and write the four finite difference equations for u at $x = 0.2$, 0.4, 0.6, and 0.8.

You will notice that the matrix constructed in Problem 1 has nonzeros on only three bands including the main diagonal; all other matrix elements are zero. The full matrix requires $(M-2)^2$ storage locations, but, if we're careful, we can instead store all the data in $O(M)$ locations by agreeing to store only the nonzero elements, along with their row and column indices. This is a standard technique for storing *sparse matrices*, those whose elements are mostly zero.

Let's see how this finite difference method is implemented.

Problem 2.

The Matlab function `finitediff1.m` on the Web site (www.computer.org/cise/homework/) implements the finite difference method for our equation. The inputs are the parameter M and the functions `a`, `c`, and `f` that define the equation. Each of these functions takes a vector of points as input and returns a vector of function values. (The function `a` also returns a second vector of values of a'.) The outputs of `finitediff1.m` are a vector `ucomp` of computed estimates of u at the mesh points `xmesh`, along with the matrix `A` and the right-hand side `g` from which `ucomp` was computed, so that `A ucomp = g`. Add documentation to the function `finitediff1.m` so that a user could easily use it, understand the method, and modify the function if necessary.

There is a mismatch in `finitediff1.m` between our approximation to u'', which is second order in h, and our approximation to u', which is only first order. We can compute a better solution, for the same cost, by using a second-order (central difference) approximation to u', so let's make this change to our function.

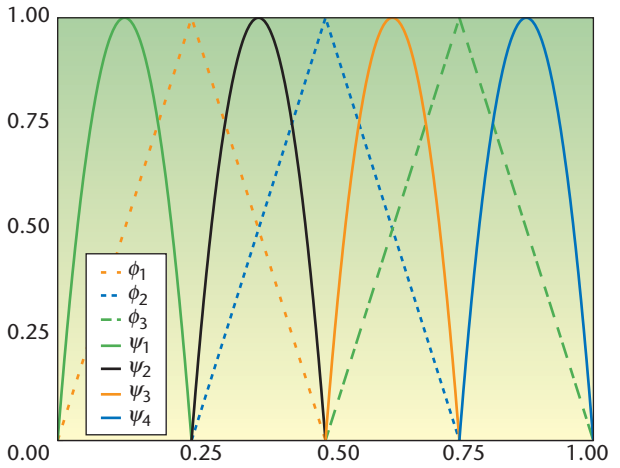

Figure 1. Hat functions. The nonzero pieces of the three linear (ϕ) and four quadratic (ψ) basis functions for three interior mesh points and four subintervals ($M = 5$).

Problem 3.

Define a central difference approximation to the first derivative by

$$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h}.$$

a. Use the Taylor series expansions

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(\xi_1),$$

$$u(x-h) = u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{6}u'''(\xi_2),$$

where ξ_1 is some point between x and $x + h$, and ξ_2 is some point between x and $x - h$, to show that the difference between $u'(x)$ and our approximation is $O(h^2)$ if u has a continuous third derivative.

b. Modify the function of Problem 2 to produce a function `finitediff2.m` that uses this approximation in place of the first-order approximation.

The Finite Element Method

We'll use a *Galerkin* approach to solving our problem with finite elements. In particular, we notice that

$$-(a(x)u'(x))' + c(x)u(x) = f(x) \text{ for } x \in (0, 1)$$

implies that

$$\int_0^1 (-(a(x)u'(x))' + c(x)u(x))v(x)dx = \int_0^1 f(x)v(x)dx$$

for all functions v. Now we use integration by parts on the

first term, recalling that our boundary values are zero, to obtain the equation

$$\int_0^1 a(x)u'(x)v'(x) + c(x)u(x)v(x)dx = \int_0^1 f(x)v(x)dx.$$

If a, c, and f are smooth functions (that is, their first few derivatives exist), then the solution to our differential equation satisfies the boundary conditions and has a first derivative, with the integral of $(u'(x))^2$ on [0, 1] finite. We call the space of all such functions H_0^1, which is also the space from which we draw v.

But how does this help us solve the differential equation? We'll first choose a subspace S_h of H_0^1 that contains functions that are good approximations to every function in H_0^1, and then we'll look for a function $u_h \in S_h$ so that

$$\int_0^1 a(x)u_h'(x)v_h'(x) + c(x)u_h(x)v_h(x)dx = \int_0^1 f(x)v_h(x)dx$$

for all functions $v_h \in S_h$. This will give us an approximate solution to our problem.

A common choice for S_h is the set of functions that are continuous and linear on each interval $[x_{j-1}, x_j] = [(j-1)h, jh]$, $j = 1, \ldots, M-1$ (piecewise linear elements), where $h = 1/(M-1)$. We can construct our solution using any basis for S_h, but one basis is particularly convenient: the set of *hat functions* $\phi_j, j = 1, \ldots, M-2$, where

$$\phi_j(x) = \begin{cases} \dfrac{x - x_{j-1}}{x_j - x_{j-1}} & x \in [x_{j-1}, x_j] \\ \dfrac{x - x_{j+1}}{x_j - x_{j+1}} & x \in [x_j, x_{j+1}]. \\ 0 & \text{otherwise} \end{cases}$$

These are designed to satisfy $\phi_j(x_j) = 1$ and $\phi_j(x_k) = 0$ if $j \neq k$ (see Figure 1 for an illustration). Note that ϕ_j is nonzero only on the interval (x_{j-1}, x_{j+1}), but it is defined everywhere.

Now we can express our approximate solution u_h as

$$u_h(x) = \sum_{j=1}^{M-2} u_j \phi_j(x)$$

for some coefficients u_j, which happen to be approximate values for $u(x_j)$.

If we define

$$a(u, v) = \int_0^1 (a(x)u'(x)v'(x) + c(x)\, u(x)v(x))dx,$$

$$(f, v) = \int_0^1 f(x)v(x)dx,$$

then our solution u satisfies

$$a(u, v) = (f, v)$$

for all $v \in H_0^1$. Next, we demand that our approximate solution $u_h \in S_h$ satisfy

$$a(u_h, v_h) = (f, v_h)$$

for all $v_h \in S_h$. In Problem 4, we reduce this to a linear system of equations that can be solved for the coefficients u_j; we implement our ideas in Problem 5.

PROBLEM 4.

a. Since the functions ϕ_j form a basis for S_h, any function $v_h \in S_h$ can be written as

$$v_h(x) = \sum_{j=1}^{M-2} v_j \phi_j(x)$$

for some coefficients v_j. Show that if

$a(u_h, \phi_j) = (f, \phi_j)$ for $j = 1, \ldots, M-2$, then

$a(u_h, v_h) = (f, v_h)$ for all $v_h \in S_h$.

b. Putting the unknowns u_j in a vector $\mathbf{u}$, we can write the resulting system of equations as $\boldsymbol{A}\mathbf{u} = \mathbf{g}$ where the (j, k) entry in $\boldsymbol{A}$ is $a(\phi_j, \phi_k)$ and the jth entry in $\mathbf{g}$ is (f, ϕ_j). Write this system of equations for $M = 6$, $a(x) = 1$, and $c(x) = 0$, and then compare with your solution to Problem 1.

PROBLEM 5.

Write a function `fe_linear.m` that has the same inputs and outputs as `finitediff1.m` but computes the finite element approximation to the solution using piecewise linear elements. Remember to store $\boldsymbol{A}$ as a sparse matrix.

It can be shown that the computed solution is within $O(h^2)$ of the exact solution, if the data is smooth enough. We can get better accuracy if we use *higher-order elements*; for example, piecewise quadratic elements would produce a result

within $O(h^3)$ for smooth data. A convenient basis for this set of elements is the piecewise linear basis plus $M-1$ quadratic functions ψ_j that are zero outside $[x_{j-1}, x_j]$ and satisfy

$$\begin{aligned} \psi_j(x_j) &= 0 \\ \psi_j(x_{j-1}) &= 0 \\ \psi_j(x_{j-1} + h/2) &= 1 \end{aligned}$$

for $j = 1, \ldots, M-1$. See Figure 1 for an illustration.

Problem 6.

Write a function `fe_quadratic.m` that has the same inputs and outputs as `finitediff1.m` but computes the finite element approximation to the solution using piecewise quadratic elements. To keep the number of unknowns comparable to the number in the previous functions, let the number of intervals be $m = \lfloor M/2 \rfloor$. When $M = 10$, for example, we have five quadratic basis functions (one for each subinterval) and four linear ones (one for each interior mesh point). If you order the basis elements as $\psi_1, \phi_1, \ldots, \psi_{m-1}, \phi_{m-1}, \psi_m$, then the matrix A will have five nonzero bands including the main diagonal. Compute one additional output `uval`, which is the finite element approximation to the solution at the $m-1$ interior mesh points and the m midpoints of each interval, where the $2m-1$ equally spaced points are ordered smallest to largest. (In our previous methods, this was equal to `ucomp`, but now the values at the midpoints of the intervals are a linear combination of the linear and quadratic elements.)

Now we have four solution algorithms, so we define a set of functions for experimentation:

$$u_1(x) = x(1-x)e^x,$$

$$u_2(x) = \begin{cases} u_1(x) & \text{if } x \le 2/3 \\ x(1-x)e^{2/3} & \text{if } x > 2/3 \end{cases}$$

$$u_3(x) = \begin{cases} u_1(x) & \text{if } x \le 2/3 \\ x(1-x) & \text{if } x > 2/3 \end{cases}$$

$$a_1(x) = 1,$$

$$a_2(x) = 1 + x^2,$$

$$a_3(x) = \begin{cases} a_2(x) & \text{if } x \le 1/3 \\ x + 7/9 & \text{if } x > 1/3 \end{cases}$$

$$c_1(x) = 0,$$

$$c_2(x) = 2,$$

$$c_3(x) = 2x.$$

For each particular choice of u, a, and c, we define f using Equation 1.

Problem 7.

Use your four algorithms to solve seven problems:

- a_1 with c_j ($j = 1, 2, 3$) and true solution u_1.
- a_j ($j = 2, 3$) with c_1 and true solution u_1.
- a_1 and c_1 with true solution u_j ($j = 2, 3$).

Compute three approximations for each algorithm and each problem, with the number of unknowns in the problem chosen to be 9, 99, and 999. For each approximation, print $\| \mathbf{u}_{\text{computed}} - \mathbf{u}_{true} \|_\infty$, where $\mathbf{u}_{true}$ is the vector of true values at the points jz, and where $z = 1/10, 1/100$, or $1/1{,}000$, respectively.

Discuss the results:

- How easy is it to program each of the four methods? Estimate how much work Matlab does to form and solve the linear systems. (The work to solve the tridiagonal systems should be about $5M$ multiplications, and the work to solve the five-diagonal systems should be about $11M$ multiplications, so you just need to estimate the work in forming each system.)
- For each problem, note the observed convergence rate r: if the error drops by a factor of 10^r when M is increased by a factor of 10, then the observed convergence rate is r.
- Explain any deviations from the theoretical convergence rate: $r = 1$ and $r = 2$ for the two finite difference implementations, and $r = 2$ and $r = 3$ for the finite element implementations when measuring $(u - u_h, u - u_h)^{1/2}$.

In doing this work, we begin to understand the complexities of implementation of finite difference and finite element methods. We have left out many features that a practical implementation should contain. In particular, the algorithm should be adaptive, estimating the error on each mesh interval and subdividing the intervals (or raising the order of polynomials) where the error is too high. And we need to handle partial differential equations, too. Luckily, there are good implementations of these methods for two- and three-dimensional domains, so we don't need to write our own.

BLIND DECONVOLUTION: A MATTER OF NORM

By Dianne P. O'Leary

WE WANT TO MINIMIZE

$$\frac{1}{2}\|\boldsymbol{E}\|_F^2 + \frac{1}{2}\|\mathbf{r}\|_2^2, \tag{1}$$

where $\mathbf{r} = \mathbf{g} - (\boldsymbol{K} + \boldsymbol{E})\mathbf{f}$, and $\boldsymbol{K}$ and $\boldsymbol{E}$ are Toeplitz matrices.

PROBLEM 1.

Show that $\boldsymbol{E}\mathbf{f}$ can be written as $\boldsymbol{F}\hat{\mathbf{e}}$, where $\hat{\mathbf{e}}$ is the vector that has entries $\hat{e}_i$, and $\boldsymbol{F}$ is a matrix whose entries depend on the entries in the vector $\mathbf{f}$. In other words, find a matrix $\boldsymbol{F}$ so that $\boldsymbol{E}\mathbf{f} = \boldsymbol{F}\hat{\mathbf{e}}$.

Answer:

Writing out the expression $\boldsymbol{E}\mathbf{f}$ component by component and, for each component, solving for the ith row of $\boldsymbol{F}$, to make $(\boldsymbol{E}\mathbf{f})_i$ equal to that row times $\hat{\mathbf{e}}$, we find that $\boldsymbol{F}$ is a Toeplitz matrix of size $m \times (m + n - 1)$ with first row equal to $[f_n, f_{n-1}, \ldots, f_1, 0, \ldots, 0]$ and first column equal to $[f_n, 0, \ldots, 0]$.

PROBLEM 2.

Derive the Newton direction for Equation 1. To do this, use the definitions of $\boldsymbol{E}$ (in terms of $\hat{\mathbf{e}}$) and $\mathbf{r}$, and then differentiate Equation 1 with respect to $\hat{\mathbf{e}}$ and $\mathbf{f}$.

Answer:

Let $\mathbf{d} = [1, \sqrt{2}, \ldots, \sqrt{n}, \ldots, \sqrt{n}, \sqrt{n-1}, \ldots, 1]$ be a vector of length $m + n - 1$ and let $\boldsymbol{D}$ be the diagonal matrix with entries $\mathbf{d}$. Then

$$F(\hat{\mathbf{e}}, \mathbf{f}) \equiv \frac{1}{2}\|\boldsymbol{E}\|_F^2 + \frac{1}{2}\|\mathbf{r}\|_2^2$$

$$= \frac{1}{2}\sum_{i=1}^{m+n-1} d_i^2 \hat{e}_i^2 + \frac{1}{2}\sum_{i=1}^{m} r_i^2$$

$$= \frac{1}{2}\sum_{i=1}^{m+n-1} d_i^2 \hat{e}_i^2 + \frac{1}{2}\sum_{i=1}^{m}\left(g_i - \sum_{j=1}^{n}(k_{ij} + e_{ij})f_j\right)^2.$$

We need the gradient and Hessian matrix of this function. Noting that $\boldsymbol{E}_{ij} = \hat{e}_{n+i-j}$, and letting $\delta_{ij} = 0$ if $i \neq j$ and 1 if $i = j$, we compute

$$\frac{\partial F(\hat{\mathbf{e}}, \mathbf{f})}{\partial \hat{e}_\ell} = d_\ell^2 \hat{e}_\ell - \sum_{i=1}^{m} r_i f_{n+i-\ell} = \left(\boldsymbol{D}^2\hat{\mathbf{e}} - \boldsymbol{F}^T\mathbf{r}\right)_\ell,$$

$$\frac{\partial F(\hat{\mathbf{e}}, \mathbf{f})}{\partial \mathbf{f}_\ell} = -\sum_{i=1}^{m} r_i\left(k_{i\ell} + \hat{e}_{n+i-\ell}\right) = -\left((\boldsymbol{K} + \boldsymbol{E})^T\mathbf{r}\right)_\ell,$$

$$\frac{\partial^2 F(\hat{\mathbf{e}}, \mathbf{f})}{\partial \hat{e}_\ell \partial \hat{e}_q} = \delta_{\ell,q} d_\ell^2 + \sum_{i=1}^{m} f_{n+i-\ell} f_{n+i-q} = \left(\boldsymbol{D}^2 + \boldsymbol{F}^T\boldsymbol{F}\right)_{\ell q},$$

$$\frac{\partial^2 F(\hat{\mathbf{e}}, \mathbf{f})}{\partial \hat{e}_\ell \partial f_q} = r_{\ell+q} + \sum_{i=1}^{m}\left(k_{iq} + e_{iq}\right) f_{n+i-\ell} = \left(\boldsymbol{R} + (\boldsymbol{K} + \boldsymbol{E})^T\boldsymbol{F}\right)_{\ell q},$$

$$\frac{\partial F(\hat{\mathbf{e}}, \mathbf{f})}{\partial \mathbf{f}_\ell \partial \mathbf{f}_q} = \sum_{i=1}^{m}\left(k_{i\ell} + e_{i\ell}\right)\left(k_{iq} + e_{iq}\right) = \left((\boldsymbol{K} + \boldsymbol{E})^T(\boldsymbol{K} + \boldsymbol{E})\right)_{\ell q},$$

where out-of-range entries in summations are assumed to be zero and $\boldsymbol{R}$ is a matrix whose nonzero entries are components of $\mathbf{r}$. So

$$g = \nabla F(\hat{\mathbf{e}}, \mathbf{f}) = \begin{bmatrix} \mathbf{D}^2\hat{\mathbf{e}} - \boldsymbol{F}\mathbf{r} \\ (\boldsymbol{K} + \boldsymbol{E})^T\mathbf{r} \end{bmatrix},$$

$$\boldsymbol{H}(\hat{\mathbf{e}}, \mathbf{f}) = \begin{bmatrix} \boldsymbol{D}^2 + \boldsymbol{F}^T\boldsymbol{F} & \boldsymbol{R}^T + \boldsymbol{F}^T(\boldsymbol{K} + \boldsymbol{E}) \\ \boldsymbol{R} + (\boldsymbol{K} + \boldsymbol{E})^T\boldsymbol{F} & (\boldsymbol{K} + \boldsymbol{E})^T(\boldsymbol{K} + \boldsymbol{E}) \end{bmatrix}.$$

PROBLEM 3.

Show that this Newton direction is approximately the same as the solution to the least squares problem

$$\min_{\Delta\hat{\mathbf{e}},\Delta\mathbf{f}} \left\| \begin{bmatrix} F & K+E \\ D & 0 \end{bmatrix} \begin{bmatrix} \Delta\hat{\mathbf{e}} \\ \Delta\mathbf{f} \end{bmatrix} + \begin{bmatrix} -\mathbf{r} \\ D\hat{\mathbf{e}} \end{bmatrix} \right\|_2 .$$

(In particular, the least squares solution is very close to the Newton direction if the model is good, so that $\|\mathbf{r}\|$ is small.)

Answer:

The least squares problem is of the form

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|^2 ,$$

where

$$\mathbf{x} = \begin{bmatrix} \Delta\hat{\mathbf{e}} \\ \Delta\mathbf{f} \end{bmatrix}$$

and A and $\mathbf{b}$ are the given matrix and vector. So to minimize $\|A\mathbf{x} - \mathbf{b}\|^2 = (A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b})$, we set the derivative equal to zero, obtaining

$$A^TA\mathbf{x} - A^T\mathbf{b} = \mathbf{0},$$

and the solution to this equation is a minimizer if the second derivative A^TA is positive definite (which requires that A have full column rank). Returning to our original notation, we get

$$\begin{bmatrix} F & K+E \\ D & 0 \end{bmatrix}^T \begin{bmatrix} F & K+E \\ D & 0 \end{bmatrix} \begin{bmatrix} \Delta\hat{\mathbf{e}} \\ \Delta\mathbf{f} \end{bmatrix} = -\begin{bmatrix} F & K+E \\ D & 0 \end{bmatrix}^T \begin{bmatrix} -\mathbf{r} \\ D\hat{\mathbf{e}} \end{bmatrix},$$

and this matches the expression $H\mathbf{p} = -\mathbf{g}$ from Problem 2 except that the matrix R (which should be small if the model is good) is omitted.

PROBLEM 4.

Write a function `[f,ehat,r,itn] = stls(K,g,lambda,tol)` that uses a variant of Newton's method to solve our Toeplitz-constrained problem in a stable and efficient way. Use the least squares problem above to compute the approximate Newton direction. Start the iteration with $\hat{\mathbf{e}} = \mathbf{0}$ and $\mathbf{f}$ equal to the least squares solution. (Starting with $\mathbf{f} = \mathbf{0}$ can cause difficulties.) Stop the iteration when the norm of the approximate Newton step is smaller than `tol`, and then set `itn` to the number of iterations. Provide documentation for your function. Use it on the data from the Web site (www.computer.org/cise/homework), setting $\lambda = 0.06$ and `tol` $= 10^{-3}$. Plot the solution, and print the residual norm, the solution norm, and the number of Newton iterations.

Answer:

See the Matlab code posted on the Web site (www.computer.org/cise/homework).

PROBLEM 5.

a. Show that when $p = 1$, minimizing

$$\left\| \begin{bmatrix} F & K+E \\ D & 0 \\ 0 & \lambda I \end{bmatrix} \begin{bmatrix} \Delta\hat{\mathbf{e}} \\ \Delta\mathbf{f} \end{bmatrix} + \begin{bmatrix} -\mathbf{r} \\ D\hat{\mathbf{e}} \\ \lambda\mathbf{f} \end{bmatrix} \right\|_p$$

over all choices of $\Delta\mathbf{f}$ and $\Delta\hat{\mathbf{e}}$ is equivalent to solving the linear programming problem

$$\min_{\Delta\hat{\mathbf{e}},\Delta\mathbf{f},\bar{\sigma}} \quad \bar{\sigma} = \sum_{i=1}^{m} \bar{\sigma}_{1_i} + \sum_{i=1}^{q} \bar{\sigma}_{2_i} + \sum_{i=1}^{n} \bar{\sigma}_{3_i}$$

subject to

$$\begin{array}{ccccc} -\bar{\sigma}_1 & \le & F\Delta\hat{\mathbf{e}} + (K+E)\Delta\mathbf{f} - \mathbf{r} & \le & \bar{\sigma}_1 \\ -\bar{\sigma}_2 & \le & D\Delta\hat{\mathbf{e}} + D\hat{\mathbf{e}} & \le & \bar{\sigma}_2 \\ -\bar{\sigma}_3 & \le & \lambda\Delta\mathbf{f} + \lambda\mathbf{f} & \le & \bar{\sigma}_3, \end{array}$$

where $\bar{\sigma}_1 \in \mathbf{R}^{m\times 1}$ and $\bar{\sigma}_2 \in \mathbf{R}^{q\times 1}$, and $\bar{\sigma}_3 \in \mathbf{R}^{n\times 1}$.

(Note that we have added a regularization component to the $p = 1$ norm from the original problem statement.)

b. Derive a similar linear program to compute $\Delta\hat{\mathbf{e}}$, $\Delta\mathbf{f}$ when $p = \infty$.

Answer:

a. Given any $\Delta\hat{\mathbf{e}}$, $\Delta\mathbf{f}$, let

$$\begin{bmatrix} \bar{\sigma}_1 \\ \bar{\sigma}_2 \\ \bar{\sigma}_3 \end{bmatrix} = \left\| \begin{bmatrix} F & K+E \\ D & 0 \\ 0 & \lambda I \end{bmatrix} \begin{bmatrix} \Delta\hat{\mathbf{e}} \\ \Delta\mathbf{f} \end{bmatrix} + \begin{bmatrix} -\mathbf{r} \\ D\hat{\mathbf{e}} \\ \lambda\mathbf{f} \end{bmatrix} \right\|.$$

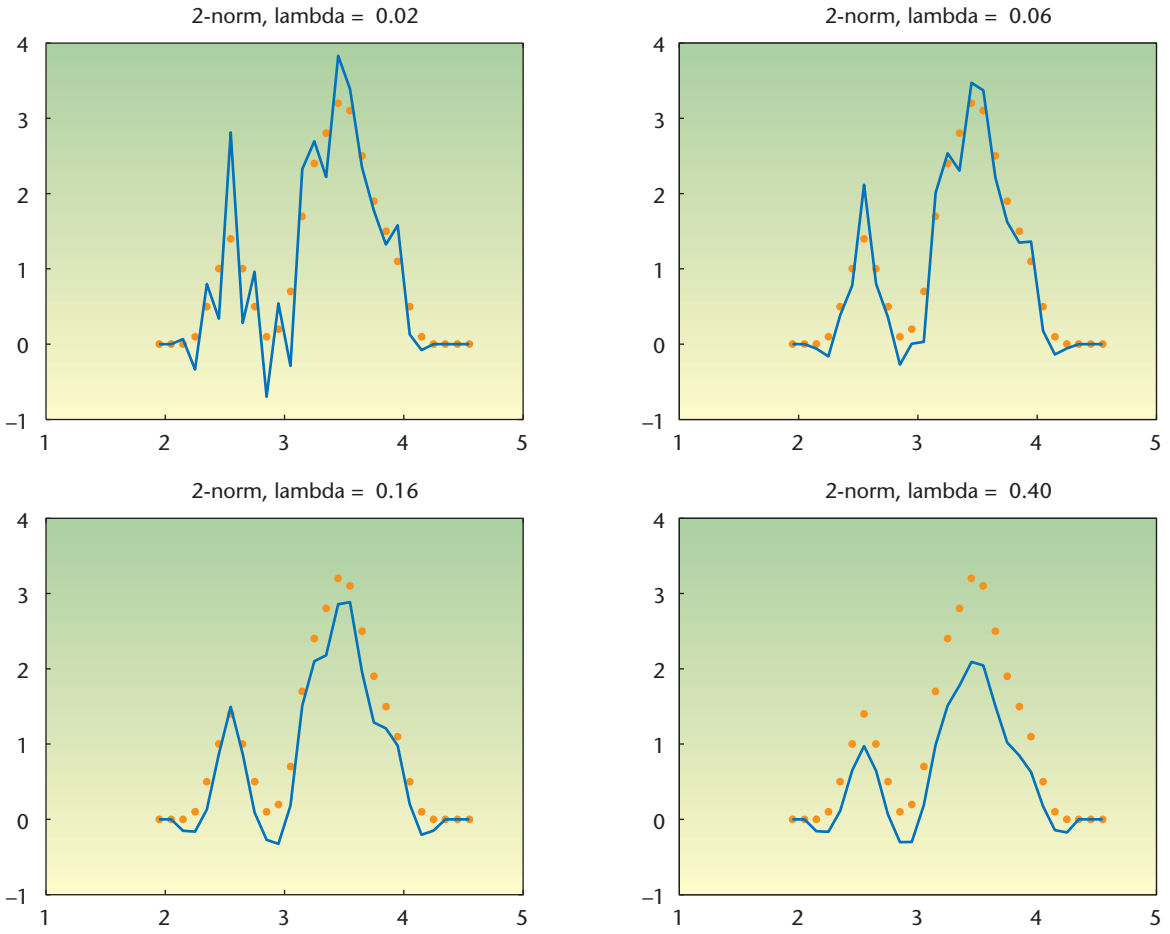

Figure A. Results from the structured total least squares (STLS) algorithm for various values of λ. The solid curves are the computed solutions, and the dotted curves represent the data.

Then $\Delta\hat{\mathbf{e}}$, $\Delta\mathbf{f}$, $\bar{\sigma}_1$, $\bar{\sigma}_2$, and $\bar{\sigma}_3$ form a feasible solution to the linear programming problem, and

$$\bar{\sigma} = \sum_{i=1}^{m} \bar{\sigma}_{1_i} + \sum_{i=1}^{q} \bar{\sigma}_{2_i} + \sum_{i=1}^{n} \bar{\sigma}_{3_i} = \left\| \begin{bmatrix} F & K+E \\ D & 0 \end{bmatrix} \begin{bmatrix} \Delta\hat{\mathbf{e}} \\ \Delta\mathbf{f} \end{bmatrix} + \begin{bmatrix} -\mathbf{r} \\ D\hat{\mathbf{e}} \end{bmatrix} \right\|_p .$$

Therefore, a solution to the linear programming problem minimizes the norm, and a minimizer of the norm is a solution to the linear programming problem, so the two are equivalent.

b. By similar reasoning, we obtain

$$\min_{\Delta\hat{\mathbf{e}},\Delta\mathbf{f},\bar{\sigma}} \quad \bar{\sigma}$$

subject to

$$\begin{array}{ccccc} -\bar{\sigma}\mathbf{1} & \leq & F\Delta\hat{\mathbf{e}} + (K+E)\Delta\mathbf{f} - \mathbf{r} & \leq & \bar{\sigma}\mathbf{1} \\ -\bar{\sigma}\mathbf{1} & \leq & D\Delta\hat{\mathbf{e}} + D\hat{\mathbf{e}} & \leq & \bar{\sigma}\mathbf{1} \\ -\bar{\sigma}\mathbf{1} & \leq & \lambda\Delta\mathbf{f} + \lambda\mathbf{f} & \leq & \bar{\sigma}\mathbf{1}, \end{array}$$

where $\mathbf{1}$ is a column vector with each entry equal to 1, and of dimension m in the first two inequalities, q in the second two, and n in the last two.

PROBLEM 6.

Write a function `[f,ehat,r,itn] = stln1(K,g, lambda,tol)` that uses a variant of Newton's method to solve the problem when $p = 1$. Use the linear program to compute an approximate Newton direction. Start the iteration with $\hat{\mathbf{e}} = \mathbf{0}$ and $\mathbf{f} = \mathbf{1}$. Stop the iteration when the norm of the approximate Newton step is smaller than `tol`, and set `itn` to the number of iterations. Use it on the data from the Web site (www.computer.org/cise/homework), setting $\lambda = 0.06$ and `tol` = 10^{-3}. Plot the solution, and print the residual norm, the solution norm, and the number of iterations.

Repeat for the case $p = \infty$.

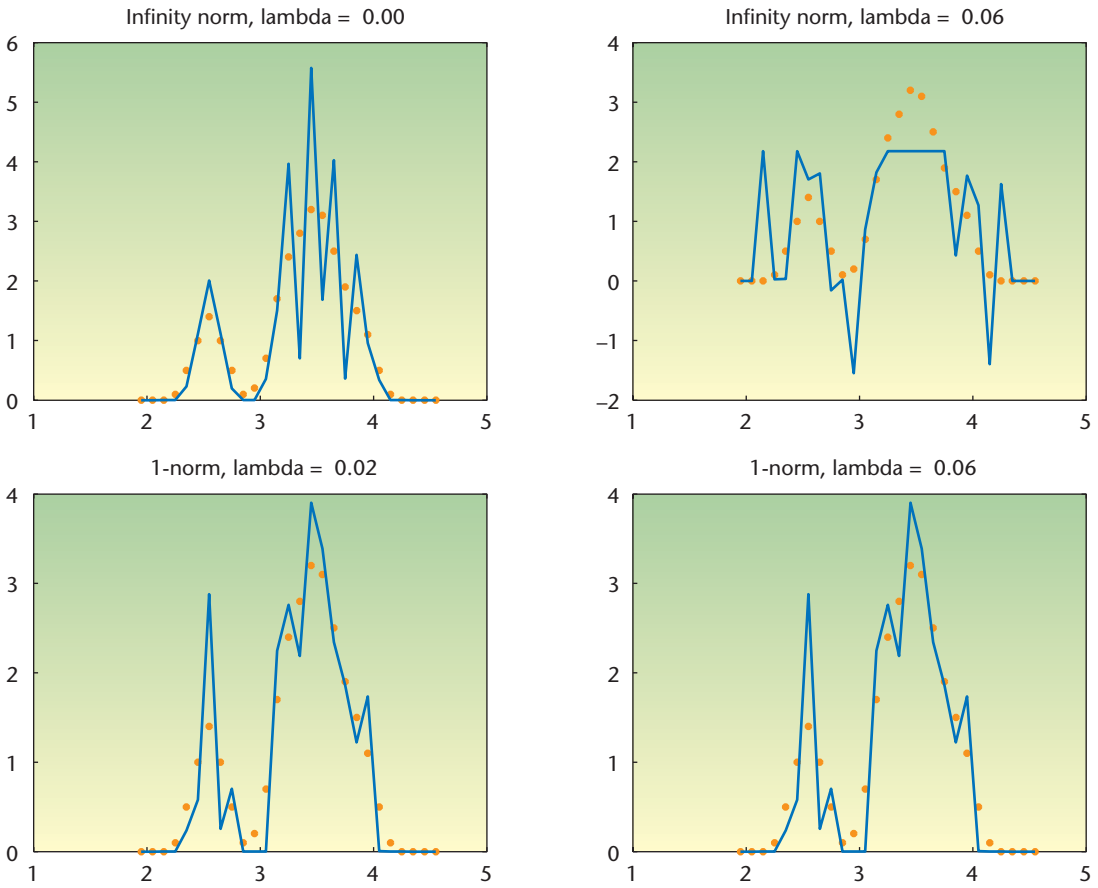

Figure B. Results from the structured total least norm (STLN) algorithm, using the 1-norm and the ∞-norm, for various values of λ.

Answer:

See the Matlab code posted at www.computer.org/cise/homework.

PROBLEM 7.

Compare the results from Problems 4 and 6 with those of the last issue by answering these two questions: How does the quality compare? How does the amount of work compare?

Answer:

Figures A and B show results for various values of λ. The estimated energy levels and counts are

Bin centers	2.55	3.25	3.55	3.85
True counts	1.00	1.50	2.00	1.00
Least squares	1.00	1.39	1.91	0.90
STLS	1.00	1.20	1.59	0.64
STLN, 1-norm	1.00	0.96	1.36	0.60

The structured total least norm (STLN) algorithm using the ∞-norm produced counts that were sometimes quite negative; nonnegativity constraints could be added to improve the results.

All the structured algorithms had a linear convergence rate, rather than the quadratic rate expected from Newton's method, because the residual **r** in this problem is large, which means the approximate Newton direction isn't very accurate.

Least squares works best on this data set, because the Toeplitz assumption used by the structured algorithms' STLS (structured total least squares) and STLN is violated by the way the data was generated. It's worthwhile to generate a new data set that satisfies this assumption, and then experiment further.

Dianne P. O'Leary is a professor of computer science and a faculty member in the Institute for Advanced Computer Studies and the Applied Mathematics Program at the University of Maryland. She received a BS in mathematics from Purdue University and a PhD in computer science from Stanford. O'Leary is a member of SIAM, ACM, and AWM. Contact her at oleary@cs.umd.edu; www.cs.umd.edu/users/oleary/.